Project 2 Report
ECE 449

Peter Chinetti

September 30, 2015

Instructor: Professor Wang

1 Feature List

The features chosen for the initial release were recognition and parsing of module names.
Module parsing was chosen for the initial release because it required the smallest modification from the original code. Behind the scenes, a class system for tokens was implemented that stored a token's type and data. This allows for easy traversal of the token list while parsing for syntax.
The final release included parsing of all features: modules, wires, and components.

2 Test Case Design

The test cases are designed to check parsing of multiple modules per file. The only feature of the initial release was module recognition, so no further test could be written.
To test full functionality of the parser, the golden test files were used. The parser had no difficulty in parsing the smaller test files, but ran into significant difficulty when working on the larger test files (cpu32, cpu8, and s15850).
The difficulty was caused by a misunderstanding of how objects were passed in C++. I had forgotten that if vectors were passed whole, not by reference, they were copied. This caused the runtime to balloon to $O(n^2)$. Tellingly, parsing a single component in the cpu32 took 0.16 seconds!.

3 Implementation: Initial vs. Final

In the final release, there were no classes for submodules (wires, components, and pins). In the final release, those classes are implemented.
Also, the original design of the final release featured a recursive implementation

of the provided Finite State Machine. This system works well for small files, but scales poorly to handle the larger test files. As discussed above this poor performance was caused by the vector copy, not the recursive stack frame overhead. However, when trying to debug the poor performance, I assumed that the recursive overhead was causing the slowness and removed it.
The final final design was a flattened, looping implementation of the parser, with the token vector passed by reference. It performs well, taking only 2.9 seconds to build and run all golden tests.